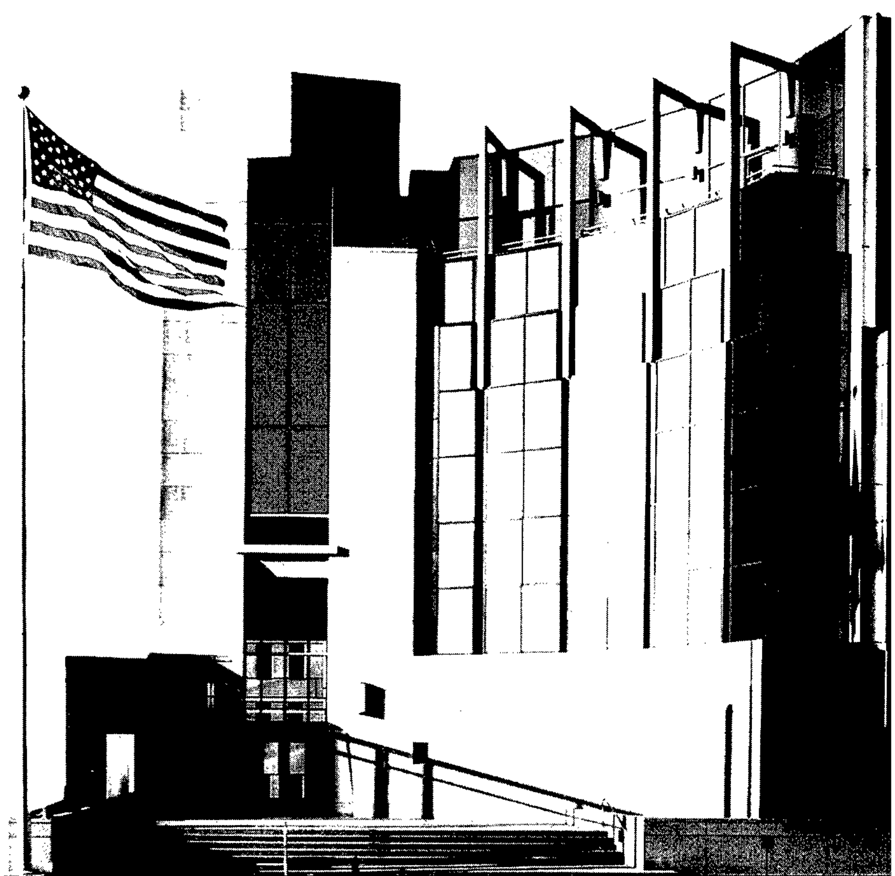
Carnegie Mellon
Software Engineering Institute

SEI Workshop on Software Architecture Representation, 16-17 January 2001

Felix Bachmann
Paul Clements
David Garlan
James Ivers
Reed Little
Robert Nord
Judy Stafford

May 2001

20011019 021



SPECIAL REPORT
CMU/SEI-2001-SR-010

DISTRIBUTION STATEMENT A
Approved for Public Release
Distribution Unlimited

Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

SEI Workshop on Software Architecture Representation, 16-17 January 2001

CMU/SEI-2001-SR-010

Felix Bachmann
Paul Clements
David Garlan
James Ivers
Reed Little
Robert Nord
Judy Stafford

May 2001

Architecture Tradeoff Analysis Initiative

This report was prepared for the

SEI Joint Program Office
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

Norton L. Compton, Lt Col., USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

List of Figures

Acknowledgements

Our sincere appreciation goes to our participants who gave their knowledge and shared their experience willingly. Thanks also go to Kurt Wallnau of the SEI who participated in our workshop and made an excellent and thought-provoking presentation on documenting component-based systems using the concept of credentials.

Abstract

To further its work in architecture-related ideas, the SEI held its first Architecture Representation Workshop, January 16-17, 2001. Five leading software architects and practitioners were invited to discuss aspects of architecture representation with senior members of the SEI technical staff. The workshop articulated best practices, identified gaps in the available technology, and set the direction for future efforts.

1 Introduction

The SEI Architecture Tradeoff Analysis initiative (SEI ATA) performs original and innovative work in architecture evaluation, attribute-based architectural styles, and other architecture-related areas. As part of this effort, the SEI created the Software Architecture Representation task to codify and extend best practices in representing and documenting software architecture. The goal is to produce a handbook to help practitioners describe a software architecture in a clear, concise, and consistent manner. Although there has been no shortage of material and literature about languages and notations that claim to do the job, we distinguish between *what* you write down and the language or notation that you *use* to capture the information.

The handbook is intended to fill this gap. It will be constructed around two axioms: First, what you document about an architecture depends upon how the information will be used. Different stakeholders require different information. For example, documentation that was designed to introduce a system will differ from documentation that was designed for an architectural evaluation. Second, documenting an architecture is primarily a matter of detailing the relevant structures or *views*, and then detailing the appropriate trans-view information. A view depicts the software architecture by documenting only certain entities and relations. Choosing the specific view to document depends, again, on its intended use and other factors. Furthermore, chosen views should complement and be consistent with each other. The entire documentation package should also include rationale, usage guidance, and other information that applies to more than one view or to the architecture as a whole.

The handbook will address both what information to include and how best to present it. The SEI took a major step toward addressing these issues through its first Software Architecture Representation Workshop, which was held January 16-17, 2001. The format of the workshop followed others held by the SEI. It was built around a small cadre of experts and attendance was by invitation only. This insured a high-bandwidth information exchange among people with first-hand knowledge of the topic. Furthermore, the participants delivered papers on their areas of expertise. (These papers are included as appendices to this report.) Following the presentations, participants were assigned to working groups that addressed particular aspects of the topic. At the conclusion of the event, the working groups reported to each other.

Following each of its workshops, the SEI produces and distributes a report to all participants. They, in turn, review it to ensure that the report does not disclose any confidential or proprietary information and that it captures the facts correctly. After they approve the contents, the SEI releases the report for community review. The following is the report from the first SEI Software Architecture Representation Workshop, January 16-17, 2001.

2 Workshop Summary

At this first Software Architecture Representation workshop, we were joined by five leading practitioners in the field:

- Christopher Dabrowski, National Institute of Standards and Technology (NIST), Gaithersburg, MD
- Rich Hilliard, ConsentCache, Inc., Littleton, MA
- Stephen B. Ornburn, GBC-Group, Inc., Marietta, GA
- Tony Thomson, IT/Warner Music Group, Burbank, CA
- Jeffrey Tyree, Capital One, Glen Allen, VA

Chris Dabrowski is involved in a project at NIST to transition the use of architecture description languages (ADLs) into government and industry. Rich Hilliard, a consultant specializing in software architecture, was technical editor for the working group that produced an IEEE standard recommended practice dealing with architecture representation and documentation [IEEE 00]. Steve Ornburn is an independent consultant with extensive experience in the field. Tony Thompson and Jeff Tyree are architects with their respective organizations.

Each participant made a short presentation.

Chris Dabrowski spoke about NIST's efforts to transition ADLs to industrial use, with the ultimate goal of ADL standardization. In his view, good documentation requires a domain-specific architecture description, and ADLs are poised to make a major contribution in this area. ADLs, for example, could help with managing different architectural views as well as tracking consistencies (and inconsistencies) among them. ADLs also could help with the effective and routine use of architectural styles. They could present documentation at different levels of detail on demand for different uses and different stakeholders' needs. The complexity of architecture specifications suggests the need for sophisticated automation. Clearly, the system that manages the architecture should be the one that manages its documentation. NIST has been working with Rapide, Meta-H, ACME, UML, and others.

Rich Hilliard reported on his work formalizing architecture documentation practice into IEEE Std 1471-2000, *Recommended Practice for Architectural Description of Software-Intensive Systems* [IEEE 00]. The standard established a framework of concepts and a vocabulary for discussing architectural issues of software systems. It does not specify the format or media for descriptions or prescribe a notation. It does, however, specify required content. The standard seeks to ensure that architecture documentation addresses all of its stakeholders' concerns. These are embraced in the concept of a *view*, which represents the entire system from the perspective of a set of concerns. A view may contain one or more models, and thus be expressed in one or more notations. The standard requires the documentation to explain any known inconsistencies between views. A *viewpoint* gives the constraints for constructing a view. The standard does not specify a set of viewpoints (and hence, does not specify a set of views). It does, however, require the architect to identify each viewpoint used by naming it, listing the stakeholders whose concerns it addresses, and presenting the source (if any) of the viewpoint, as well as the language, modeling techniques, or analytical methods employed.

Steve Ornburn shared his experiences helping organizations construct blueprints for product-line platforms with up to 1000 developers. In this world, enterprise architecture is an important factor. A picture of the enterprise architecture will show deployment and component diagrams. The picture also provides context—namely, constraints and opportunities for others to fit their applications into this product-line platform. Steve reported that, despite the popularity and usefulness of multi-view architectural documentation, he has encountered some resistance getting people to accept different diagrams with separate views. Stakeholders would rather have one diagram. These stakeholders include architects, technical leads, senior managers in IT, and business managers. Steve uses attributes to capture provided services, underlying technical layers, and software to support services. Attributes are used for communicating design to stakeholders, for analysis, and so forth. He finds layering a useful concept for explication and analysis and has refined his documents to make the layering clear.

Tony Thompson of Warner Music Group told us about a development process his organization uses to capture relevant system information. Called the DREAM Framework, it is a collection of tools and methods that shepherd the entire software engineering effort. Four views, or system perspectives, are used to capture the software architecture:

- technology view, which defines the collection of technologies comprising the solution, including staff skill mix requirements
- application view, which defines the structure and relationships of the software that embodies the business processes
- development view, which defines the environment of tools, methods, processes, and team structures necessary to develop and maintain systems
- data view, which defines the structure and models of the information assets of the organization, through databases and digital content

He also shared the artifact set used in his organization. It is similar to the consolidated set called for by IEEE/EIA standard 12207 that descends from MIL-STD-2167. While it is, admittedly, a very heavyweight artifact set, it has the virtue of being well defined. There is no ambiguity about what is required.

Jeff Tyree spoke about architecture documentation principles and practical problems. Architecture documentation, he said, must adhere to these principles:

- It must support the shared vision; that is, it must be consistent with how an organization views architecture and its role(s).
- It must support the organization's communication channels and various stakeholders' needs.
- It must be compatible with supporting tools. These tools are necessary to help the documentation evolve over time and to make it accessible to stakeholders throughout the organization.
- It must support the architectural process in use. In particular, it must support all the work products called for by the organization's chosen process. For example, if you are using Rational's Unified Process, the documentation must support the 4+1 views it requires.

In addition, documentation must

- follow the seven rules for good documentation given in *Software Architecture Documentation in Practice: Documenting Architectural Layers* [Bachmann 00]
- contain enough precise and detailed information to support actual construction
- support reasoning about the architecture
- support system qualities and account for a "changing of the guard" should the original architect leave

Finally, Kurt Wallnau of the SEI delivered a presentation on documenting component-based systems. He introduced the concept of a *blackboard*, a way to document properties of a component ensemble. Adopting Mary Shaw's idea of a *credential*, he presented a scheme for documenting systems in which all information is not available—the standard situation in component-based software engineering. A credential is a 3-tuple: <property, value, knowledge>, where the third element explains how we know (and with what confidence) that the second is, in fact, the correct value of the first. Credentials apply to components, or ensembles of components, that are regarded as a unit. The idea is to sum up what we know about components to draw conclusions (with a known confidence and source) about ensembles of components, and so on, until we can gain knowledge about a system.

After the presentations, the workshop divided into two groups. One group discussed the special needs that enterprise and product line architectures bring to the documentation table. The second addressed how best to promulgate and disseminate architecture documentation throughout an organization. The reports of the two working groups follow.

3 Report of Working Group #1

3.1 Working Group Goals

Working Group #1 addressed three issues related to identifying the nature of good architectural documentation. Specifically, the working group members attempted to

1. Define an architectural documentation strategy to describe product level architecture (PLA) and enterprise level architecture (ELA).
2. Determine what information is required.
3. Identify those that the information serves.

3.2 ELA and PLA: Similarities and Differences

The group first explored the similarities and differences between PLAs and ELAs. Both types are concerned with families of applications and identifying commonalities among applications. The group observed that differences between the two center on perspective. The ELA supports creating a family of applications for use in a single enterprise, while a PLA supports creating a family of applications that serve a variety of consumers. Another difference: the ELA results from focusing on the internal needs of an enterprise, while PLA is the result of focusing on external consumers. From a different perspective, the major difference is one of "constraint versus generate." In the first case, a set of applications exists within an enterprise. An enterprise architect recognizes that it would be more cost effective to design a reference architecture for sets of applications that exhibit some amount of commonality. The motivation is to reduce the cost of developing, using, and maintaining the applications. The reference architecture is then used to constrain the architectures for the related applications. In the case of product-line systems, the goal is to create an architecture that enables many product variations to be built using the same basic architecture.

3.3 ELA vs. ALA

Next, Working Group #1 identified what types of information should be documented to support creating ELA-conforming applications.

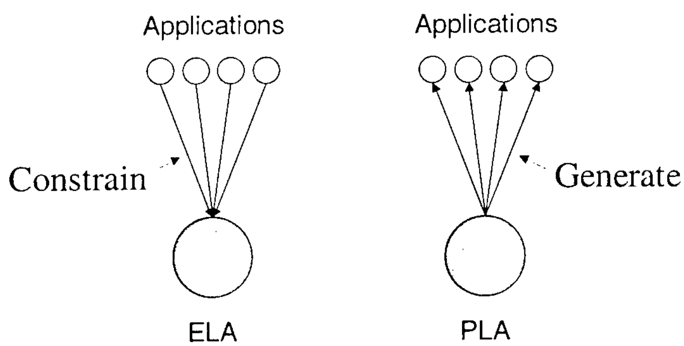


Figure 1: Relationship of ELA and PLA to Applications

The group recognized that ELAs and application level architectures (ALAs) share many relationships. The simplest of these is the one-to-many relationship; one ELA is used for many ALAs. In a somewhat more interesting variation, an ELA may itself be a product for an enterprise and may become an application that warrants constructing a PLA. Additionally, many types of power relationships exist within corporate cultures that affect the design of an ELA. The group suggested a metaphor using federal, confederate, parliamentary, dictatorial, theocratic, and anarchic models. In the federal model, the ELA has power over the ALA but the ALA retains control over many decisions. In the confederate model, the ELA is weak and the rules of architecting are decided on an application-by-application basis. The working group members readily agreed that the choice of model is based heavily on the culture of the organization, and that this fact supports creating and retaining the appropriate reference architecture.

3.4 ELA Consumers

At this point, the group turned its attention toward identifying stakeholders and, in particular, a document's target stakeholder. There was consensus that understanding and documenting the intended audience is very important. Potential stakeholders include

- capacity planners
- production support planners
- developers
- marketing strategists
- procurement specialists
- application architects
- data architects
- business process modelers

The ALA document is intended for the applications architect.

3.5 ELA Description

The following contains the outline for a description of an ELA. A completed document should help an applications architect to satisfy the criteria defined at the enterprise level. It should also clearly identify where points of variability exist and what procedures should be followed when the architect wishes to deviate from the plan. The outline that we have included here assumes that the organization uses a federal model type of power structure. We have called our document "Proposed eStandards for the Application Architect," and the working group report concluded with its table of contents.

1. Rules of Engagement
    a. Conformance criteria
    b. Standard work products
    c. Guidelines for reviewing application architectures
    d. Waivers & amendment process
2. Wiring Diagrams
    a. "As is" Wiring Diagram
    b. "To be" Wiring Diagram
3. Technical Infrastructure Standards
    a. Deployment environment
    b. For each "flex point," what options exist, when are they selected, why is the choice provided, how you do it (design heuristics, samples, etc.)
    c. Development environment
    d. Reuse commitments: components, frameworks, patterns, and shared services
    e. External standards: industry, regulatory, legal, and environmental
4. Model of Application-Independent Required Components
    a. Component-Specific Product Selections
    b. Component-Specific Technology Selections
5. Flexpoints of Architecture
6. [PLA Infrastructure and Flexpoints]

4 Report of Working Group #2

4.1 Working Group Goals

This working group tried to determine how best to instill a coherent, consistent architectural vision within an organization. In other words, how can one ensure that the overall concepts within the architecture documentation package are thoroughly and effectively disseminated throughout the organization?

The group realized that this was a two-sided question. One side hinged on the organization, its structure, its culture, its past experience with architecture, its degree of maturity with respect to concepts such as design or abstraction, and its process infrastructure. Addressing those issues, while important, seemed well beyond the scope of the working group.

Therefore, Working Group #2 focused on the second side of this question: What qualities should the architectural documentation package have so that it can be disseminated as thoroughly and effectively as possible? This question seemed to align precisely within the group's scope.

4.2 Experience

Some of our participants shared the kind of documents produced in their organizations. One organization, for example, produces a "project document." It describes the scope, organizational structure, key milestones, and related projects and groups. It also includes a risk matrix, ground rules, and assumptions. This document provides the means to track the work breakdown structure. Its goal is to attain buy-in from key stakeholders (the ones who control the funding) by convincing them that the project is on solid ground in terms of planning and management. A document like this may impose constraints on the architecture or on other documents. For example, the architecture may require a phased approach to meet stakeholders' deadlines.

Another kind of document that surfaced was the "operational concept document." Typically, it contains broad requirements and a general description of the user interface. This presents the system as seen by its end users. Besides end users, stakeholders for this kind of document include business analysts who judge whether the system as described will help the developing organization meet its business goals.

A third type of document contains detailed requirements, including low-level use cases and broad architectural constraints, as well as subsystems and configuration items (architectural components). In this " system requirements document," any description of a configuration item must contain a description of behavior and constraints. Stakeholders for this document include architects and technical leads.

The participants mentioned other documents as well. The point, however, was not to enumerate an exhaustive list, nor to discuss the merits or non-merits of any particular document. Rather, the discussion revealed some important points for the topic of architecture documentation as a whole. First, it reinforced our conviction that architecture documentation speaks to a variety of stakeholders who are interested in different aspects of the system. Second, it revealed desired documentation qualities. Some of these are addressed in the next section.

4.3 Qualities of Interest

What documentation properties will help us produce work that will be disseminated and maintained throughout an organization? A number of criteria emerged from our discussions:

- There must be a clear stakeholder target.
- Architecture must be traceable to requirements.
- Documents must capture constraints and behavior of components, of ensembles of components, and of ensembles of ensembles.
- Because systems can be divided into subsets (in the sense of being composed of ensembles), documentation should follow suit.
- Documentation should differentiate between "as designed" and "as built." As we will see shortly, even "as designed" has several shades of meaning that should be defined.
- There should be a clear stopping point, so that both the author and the reader know when the subject has been appropriately covered.
- The documentation must capture rationale, heuristics, and design knowledge.
- It must be readable and manageable. A rule of thumb is that the document should only run between 25-50 pages.

4.4 What the Meaning of "Is" Is

Documentation in general, and software architecture documentation in particular, contains many assertions. They include what components are covered, how a component works, and what relationships exist among components. There are also assertions about why a design satisfies its requirements, what will change in the future, and, for product line architectures, what must be changed to get a product-ready instance of the architecture. Furthermore, there are assertions about who wrote the documentation, when it was written, and where you can find information. You can think of an architecture document as a package of undiluted assertions. In practice, however, not all assertions are created equal.

Information coming to the architect has various pedigrees. For example, the information may represent constraints, heuristics, or simply properties.

To this, the architect adds a touch of "assertive freedom." Some of what the architect writes are facts, such as properties. Some are requirements or constraints, and no deviation is allowed. Some are non-binding decisions, suggestions, if you will. Some are placeholders, which comprise a class unto itself. Some placeholders are clearly marked TBD, but others show desired or possible values. For example, the architect may want to use a particular vendor's component, but if the product is unavailable at the time of production, something else must be substituted.

High-quality documentation should address this insidious ambiguity by clarifying the value and nature of each assertion.

4.5 Maintenance and the Role of Tools

Our group briefly discussed the role of tools in the development and maintenance of architecture documentation. This subject could rightfully occupy an entire conference. Nevertheless, we felt it important to state a few guidelines:

- Ideally, the same tool used to manage the architecture should be used to manage the architecture documentation.
- At the same time, it is not realistic to expect any tool to provide all the documentation. For example, a business unit vice president wants a few viewgraphs—what architecture tool prepares viewgraphs?
- Existing tools primarily rely on simple annotation features to help architects track a myriad of different kinds of information. While better than nothing—you can, theoretically, tag each annotation with an annotation type and write scripts to pull out annotations of a certain type—its effective use requires pre-planning, a conscientious effort, and extraordinary discipline.
- Navigation and layout problems should not be overlooked. People can only assimilate so much information on a screen, before preferring a printed copy. We are not yet in the age of paperless architecture.

4.6 Validating Architecture Documentation

Producing high-quality architecture documentation is one thing, but how will we know if it is being maintained? Documentation naturally tends to degrade over time. To address this issue, our working group outlined a review/validation procedure for architecture documentation. It is a short questionnaire that could accompany an architecture documentation package:

1. Are the document's stakeholders identified? To whom is it addressed? Are architecturally relevant concerns addressed?
2. Who is missing from the answer to #1?

3. Is every concern addressed by one or more views? Can questions and concerns be answered by the architecture description? "Concerns" should include behavior.
4. Are cross-view relations identified and described? Are consistencies across the views identified? Are inconsistencies highlighted and justified?
5. Are assertions identified as facts, heuristics, properties, requirements, non-binding decisions, desires, ranges of possibilities, placeholders, etc.?
6. Is the rationale adequately captured? For example, are areas of change explained? Are traces to requirements included?
7. Does the document explain how to exercise variabilities?
8. Is there needless or harmful redundancy?
9. Can you answer a specific question quickly? In other words, is it organized for lookup (according to who you are) and is the right information there? Ideally, the documentation package should provide a table of contents or at least a guide for each type of reader.
10. Does the documentation over-constrain or contain extraneous information?
11. Does it follow "guidelines for good documentation" given in *Software Architecture Documentation in Practice: Documenting Architectural Layers* [Bachmann 00]?
12. Is the document manageable? Ideally, each stakeholder's document should run 25-50 pages in length. This implies the documentation can easily be divided.

To these questions, we added two more:

1. Where can I find the information that relates to each question above?
2. Have all the TBDs been resolved?

4.7 A Skeletal Active Design Review for Architecture

Our group adopted the "Active Design Review" technique as the model for our hypothetical validation instrument [Parnas 85]. An active design review shuns the traditional all-hands review meeting. Instead, reviewers fill out a questionnaire about various parts of documentation. To answer the questionnaire, the reviewers must actually use the documentation, not just page through it. If the reviewers are able to provide the information, the documentation qualifies for its intended use.

What follows is the beginning of an active design review questionnaire for a hypothetical package of software architecture documentation.

1. Who are all the stakeholders for whom this documentation was written? For each stakeholder, what architectural concerns are addressed? How do you know? (e.g., point to the appropriate documentation).
2. Which stakeholder views are missing from the documentation?
3. What views are provided? For each view, where is its view type definition supplied? Where are the conditions and rationale given? (conformance to view type, refinement, etc.) Where are consistencies and inconsistencies across views explained?

4. Is every concern addressed by one or more views? Concerns are phrased as questions. Can the architecture description answer each question? Which views address each concern from #1?
5. Where are cross-view relations identified and described? Where are the conditions and rationale for consistencies and inconsistencies given?
6. Are assertions identified as constraints, heuristics, properties, facts, (derived) requirements (binding on downstream developers), non-binding decisions, desires, ranges of possibilities, placeholders, etc.? What is the primary architecture information for each view? What is your source? For each view, where are the constraints, heuristics, and properties behind the architecture information identified? For each view, where is the architecture information distinguished as fact, requirement, etc.? (Architecture information refers to the information the architect puts in the documentation, e.g., stakeholder concerns, design decisions, etc.)
7. Is the rationale adequately captured? Where are areas of change explained? Are traces to requirements included? Look for rejected alternatives, how key drivers (concerns) are addressed, selection criteria for COTS components, risks, and implied issues.
8. Does the documentation explain how to exercise variabilities? For each view, what variation points are defined and what mechanisms are employed?
9. Is there needless or harmful redundancy? Are two terms introduced that mean the same thing? Is one term used to mean two different things? Can each view be derived from or joined to another view? Explain any purposeful redundancy.
10. Can you answer a specific question quickly? Is it organized for lookup (according to who you are)? Is the right information there? Which sections of the documentation are applicable to each stakeholder? For a given stakeholder, name some concerns and write down every place where the answer can be found. Look up a specific component and list every place it is mentioned. Based on your reading, sketch the salient features of this view. (Answers should be compared for consistency.)
11. Does the documentation over constrain or contain extraneous information?
12. Does it follow the "Guidelines for Good Documentation" found in *Software Architecture Documentation in Practice: Documenting Architectural Layers* [Bachmann 00]?
13. Is the documentation manageable for each stakeholder?
14. Is the information complete? Are interfaces defined fully? Is there standard organization throughout?

5 Conclusion

The first Software Architecture Representation Workshop proved to be a high-quality focused forum for addressing some thorny issues of software architecture documentation. The outline produced by Working Group #1 and the skeletal active design review produced by Working Group #2 will both serve to further the work toward a useful handbook of software architecture documentation.

Finally, it has become a custom at these SEI workshops to discover a phrase or aphorism that expresses the essence of a relevant idea in a new and pithy way. We never have to look for these phrases; they pop out entirely on their own. This workshop was no different, and the award for best aphorism goes to Tony Thompson who noted that we've all learned for years that divide-and-conquer is the approach to system building. "In this age of component-based systems," he said, "it's time we learned to integrate and conquer."

References

[Bachmann 00] Bachmann, B.; Bass, L.; Carriere, J.; Clements, P.; Garlan, D.; Ivers, J.; Nord, R. & Little, R. *Software Architecture Documentation in Practice: Documenting Architectural Layers* (CMU/SEI-2000-SR-004, ADA 377988) Available: WWW: <URL: http://www.sei.cmu.edu/publications/documents/00.reports/00sr004.html> (2000).

[Parnas 85] Parnas, D. & Weiss, D. "Active Design Reviews: Principles and Practices," 132-6, *Proceedings of the Eighth International Conference on Software Engineering,* London, England, August, 1985.

[IEEE 00] IEEE Std. 1471–2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems* (2000).

Appendix A: Position Paper for SEI Software Architecture Documentation Workshop January 16-17, 2001

Chris Dabrowski
National Institute of Standards and Technology
January 2, 2001

A.1 Introduction

One of the potential benefits of describing software architecture is the ability to provide greater clarity and understanding than what is possible in program code. The concise representation of the essential provides a basis for communicating system design. This serves as documentation for different stakeholders and participants in the aspects of the functional components of a system, their connections and interactions, and their behavior system design process, including system analysts, designers, implementers, maintainers, and managers.

In current software practice, the development of comprehensive documentation of any aspect of a system—including its architecture—is often lengthy and tedious. This is particularly the case when describing a system using a terminology familiar to customers or when it is necessary to provide alternative views of a system to different stakeholders. To provide the greatest benefit with the least amount of effort, it should be possible for an architecture description to be stated completely in a specification created using the ADL. That is, a specification written in an ADL should be as self-documenting as possible. While additional text will always be required to provide context and design rationale, the actual specification of software architecture in the ADL should be definitive enough not to require large amounts of additional explanation and comments. For this reason, it is helpful to discuss good characteristics of architectures in the context of ADL features.

A.2 Characteristics for Good Architecture Documentation

A.2.1 Domain-Specific Vocabulary and Syntax

It should be possible to rename commonly used architectural constructs such as interfaces, components, connectors or modules to names familiar in domain. In developing a *Rapide* [1] prototype specification for the NIST Real-Time Control System (RCS) [2], the use of familiar names for types of architectural objects proved to be a significant aid in communicating understanding of the system design to RCS domain experts [3]. The same benefit may be obtained by modifying language syntax to be more familiar to domain practitioners, especially with respect to system behavior. For instance, the following examples show a portion of the RCS specification using *Rapide* vocabulary and syntax.

```
TYPE Job_Assignor IS INTERFACE;
ACTION
 OUT
   Schedule_Job (Job : Task_Command_Frame),
   Fetch_task_frame (Job : Task_Command_Frame);
 IN
   Do_task (Job : Task_Command_Frame);

BEHAVIOR

BEGIN

(?Job : Task_Command_Frame)
Do_Task (?Job) ||>
  Fetch_task_frame (?Job);;
................
```

While the above does provide a precise statement of the architecture specification, it still requires explanation of the meaning of individual constructs to domain experts not having extensive background in computer science. The specification could require less explanation by substituting more familiar terms such as a domain-specific identifier for INTERFACE, the term supplied by the ADL. This is accomplished in *Rapide*, ACME [4], and other ADLs by creating subclasses of interface types. However for purposes of understanding, it may also be desirable to substitute the term MESSAGES for ACTION, SENDS for OUT, RECEIVES for IN, and so on. Using keywords such as IF.THEN, SEND sequence may also more easily convey behavioral semantics to domain practitioners and allow the architecture specification to more directly serve as documentation. The modification of language keywords and syntax to be domain specific is an area where further research may be of benefit.

A.2.2 Organization of Specification for better understandability

The organization of the specification of architecture can affect its understandability. Documentation of architectures would be improved by permitting alternative organizations. For instance, it may be desirable to present an architecture specification as a top-down functional decomposition with its most general architectural elements first followed by lower-level subcomponents. In other circumstances, bottom-up, inside out presentations may be more appropriate—or an architecture specification may be organized by classifying modules by function type. (Tool support would be required to store a specification and present alternative organizations on demand.) These different organizations of an architecture may be combined with the layered architectural documentation approach [5] to more effectively show overall system structure[1].

It is also desirable to show alternative views of the architecture as described in [6]. It should be possible to document separate logical and process views of the architecture for presentation to stakeholders with different perspectives. It may also be necessary to combine perspectives either having separate views each of which contain a functional decomposition or a single decomposition whose components have separate views.

The use of architectural styles [7] allows better definition of particular kinds of architecture organizations, such as top-down, that are familiar in a domain. Styles help ensure that a particular kind of structure is maintained as the system evolves.

A.2.3 Levels of Abstraction

For documentation to provide complete system understanding, it should be possible to provide the same architectural specification at different levels of abstraction. A high level of abstraction is necessary for stakeholders that are not computer specialists or implementers. More detailed levels are necessary for system analysts, implementers and system maintainers. Together with support tools (see below), it should be possible to hide detail where necessary and generate versions of specifications exclude statements that are necessary for compilation or execution of simulations. Certain aspects of the logical content of the specification should be controllable as well, as for instance the ability to limit the specification to system structure alone, to limit presentation of certain views, or to eliminate lower levels of functional decomposition.

1 We have tried Bachmann's layered approach in another software architecture project with positive results. Although it is still a preliminary effort, I will try to bring an example of this to the workshop if it can be made ready in time.

A.2.4 Representation of Connections

There are also specific ADL features that help provide the proper level of abstraction for a particular purpose. Support for connectors as first-class objects would allow links between modules to be specified declaratively, making it clearer how modules are linked. First-class connectors allow particular types of modules to be connected by types of connectors, thus making system structure better defined. Related to this is the description of messages or events passing between modules in higher and lower levels of architectures. This description needs to be concise, clear and should not encumber the specification. It is perhaps desirable to consider some form of default connection in which a containing module is assumed to send and receive the messages of its components. Further research is necessary to determine language constructs that better support documentation. However, to fully realize the value of these constructs in communicating system description, tool support is necessary.

A.3 Tool Support for Self-Documentation

Good documentation requires tool support. Without it, real-world ADL specifications are too large and complex to allow good documentation to be produced quickly and efficiently. Tools will be needed to allow specification developers to easily define domain-specific grammars and to efficiently structure ADL specifications for documentation purposes. Tool support will also be needed to implement language features that allow users to see ADL specifications according to different views or at different levels of abstraction, focusing on information appropriate for a particular stakeholder and hiding unnecessary detail.

To document an ADL specification and communicate understanding of a system requires that readers be able to view individual components in isolation as well as see their logical links to other parts of the specification. Tools that allow navigation between related parts of a specification will greatly aid in this. One can imagine a graphic representation of an architecture using a layered architectural documentation approach [5] that shows overall system structure. Readers may be allowed to navigate between different components represented in the diagram in order to view the connections.

A.4 References

[1] Luckham, D. "Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial Ordering of Events," http://anna.stanford.edu/rapide, August 1996.

[2] Albus, J. S. "4-D/RCS: A Reference Model Architecture Demo III." *NISTIR 5994*, National Institute of Standards and Technology, Gaithersburg, MD, 1997.

[3] Dabrowski, C., Huang, H., Messina, E., and Horst, J. "Formalizing the NIST 4-D/RCS Reference Model Architecture Using an Architecture Description Language," *NIST Internal Report 6443*, December 1999.

[4] Garlan, D, Monroe, R., and Wile, D., "Acme: An Architecture Description Interchange Language", *Proceedings of CASCON '97*, November 1997.

[5] Bachmann, F. et. al., *Software Architecture Documentation in Practice: Documenting Architectural Layers*, CMU/SEI-2000-SR-004, December 1999.

[6] Kruchten, P. "The 4+1 View Model of Architecture", *IEEE Software*, November, 1995, pp. 42-50.

[7] Shaw, M. "Comparing Architectural Design Styles," *IEEE Software*, November, 1994, pp. 27-41.

Appendix B: IEEE Std 1471 and Beyond

Rich Hilliard
ConsentCache, Inc.
rh@ConsentCache.com

B.1 Overview

I describe the key contributions of IEEE Std 1471 to the discipline of software architecture representation. After reviewing the contributions of IEEE 1471, I discuss how we (the community interested in Software Architecture) may build upon the foundation provided by IEEE 1471 to continue to improve and disseminate techniques for architectural description.

(Although three pages is insufficient to give a useful example of an IEEE 1471-conformant architectural description, there are a number of applications of IEEE 1471 in the literature. Visit the IEEE Architecture Working Group web site (http://www.pithecanthropus.com/~awg) for links.)

B.2 IEEE Std 1471 ...

IEEE Std 1471-2000 is IEEE's *Recommended Practice for Architectural Description of Software-Intensive Systems* [7]. To my knowledge, this is the first formal standard to address what is an architectural description (AD). It was developed by the IEEE Architecture Working Group with representation from industry, other standards bodies, and academe, and was subject to intensive reviews by over 150 international reviewers, before its publication this past Fall.

IEEE 1471 establishes a set of *content requirements* on an architectural description (AD)—a collection of products to document an architecture. As such, the Standard plants a stake on how ADs should be organized, and their information content, while: (1) abstracting away from specific media (text, HTML, XML); (2) being method-neutral (it is being used with a variety of existing and new architectural methods and techniques); and (3) being notation-independent, recognizing that many diverse notations are needed for recording various aspects of architectures.

It achieves this by being based upon a conceptual framework for architectural description. (See Figure 1.) The breadth of this framework is worth appreciating relative to current work in architectural research and practice. To my mind, much of this work has focused on what are portrayed as Models in the conceptual framework, including architectural description languages, and related tools. While important, much of this work lacks a larger context needed in most practical, industrial strength applications. By reifying notions like Stakeholders and Concerns, the IEEE 1471 framework suggests a basis for dealing with these wider issues in a theory of architectural description.

B.3 Content Requirements on ADs

The content requirements of IEEE 1471 are stated in the terminology of the conceptual framework. These requirements define what it means for an architectural description (AD) to conform to the Standard. The principles underlying these requirements are briefly summarized here:

ADs are interest-relative. The audiences are the various stakeholders of the system, each with specific concerns (such as security, performance, constructability) for the architecture. An AD should be explicit in addressing these stakeholders. Therefore, an AD must explicitly identify the system's stakeholders and their concerns for the system.

Concerns form the basis for completeness. An AD must addresses all stakeholders' concerns. If it does not, it is, by definition, incomplete.

Multiple views. An AD is organized into one or more views. Each view is a representation of the entire system of interest intended to address a particular set of stakeholder concerns.

Although the use of views is hardly new with IEEE 1471, its contribution is to motivate the use of views (the source of much hand-waving in the Software Architecture literature) with respect to addressing specific concerns of specific stakeholders.

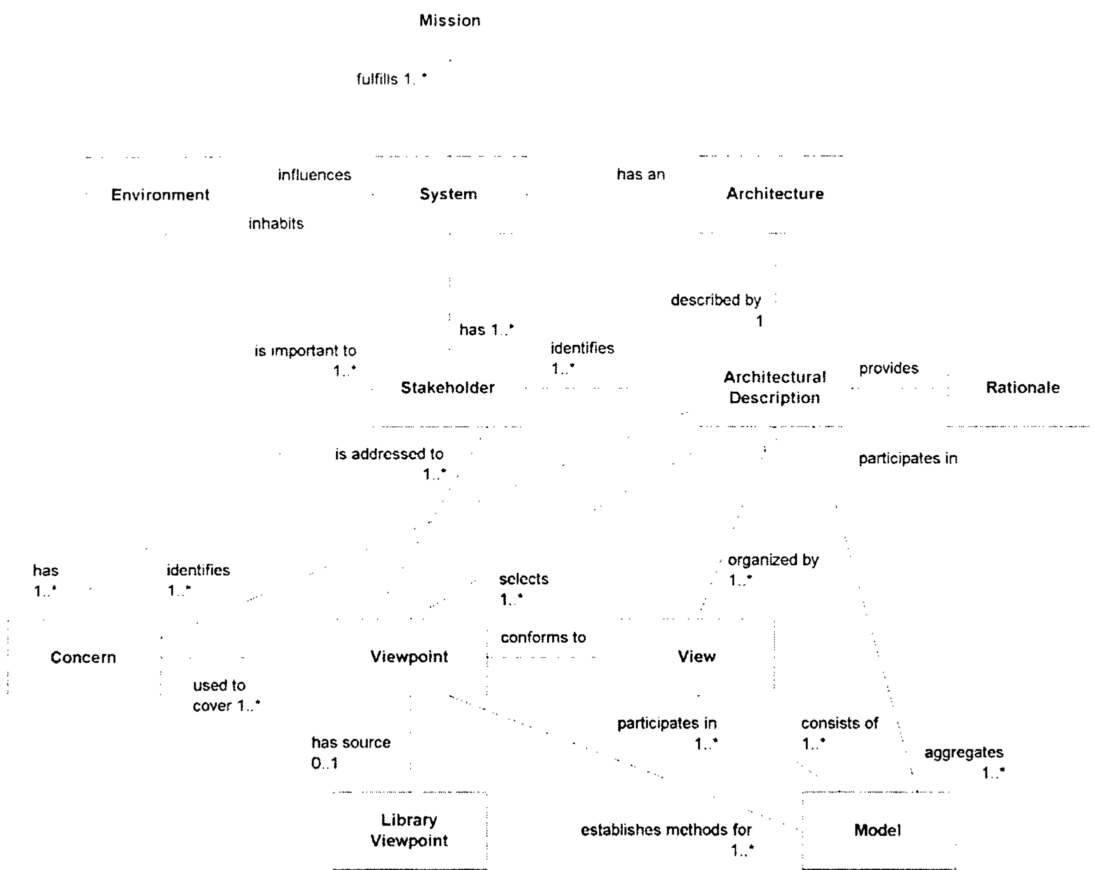


Figure B-1: IEEE 1471 Conceptual Framework

Views are modular. A view may consist of one or more architectural models. To satisfy the concerns to be addressed by a particular view, multiple notations may be used. This is one of the several places where IEEE 1471 is "parameterized" to accommodate the wide range of best practices in Software Architecture modeling.

Inter-view consistency. An AD must document any known inconsistencies among the views it contains. This is a fairly weak requirement—based on current consensus; I imagine as a community we can do much better in the future (see below).

Views are well-formed. Each view has an underlying viewpoint identifying a set of architectural concerns and specifying how the architectural description meets those concerns, using languages and notations, models, analytical techniques, and methods. A *viewpoint* is a set of conventions for constructing, interpreting and analyzing a view.

This is another "parameter" in IEEE 1471. Organizations may define and select their own set of useful viewpoints. In fact, IEEE 1471 does not even specify a fixed set of viewpoints; the

Standard is "agnostic" about where viewpoints come from. Instead, the following principle is employed:

Concerns drive viewpoint selection. Each identified stakeholder concern must be addressed by one of the selected viewpoints.

Viewpoints are first-class. Each viewpoint used in an AD is "declared" before use (either "in line" or by reference). A viewpoint declaration establishes the stakeholders addressed by the viewpoint; the stakeholder concerns to be addressed by the viewpoint; the viewpoint language, modeling techniques, or analytical methods used therein; and the source, if any, of the viewpoint ("prior art"). A viewpoint may also include: any consistency or completeness checks associated with the underlying method to be applied to the models within the view; any evaluation or analysis techniques to be applied to models within the view; and any heuristics, patterns, or other guidelines which aid in the synthesis of an associated view or its models.

This principle is perhaps the primary contribution of IEEE 1471—to provide a means by which the many architectural techniques in use today may be uniformly described so that they may be used by others, compared, and combined.

B.4 And Beyond

In addition to codifying best practices in architectural description, a goal of the IEEE for the development of IEEE 1471 was to provide a foundation for the continuing evolution of the discipline of Software Architecture. To conclude this position paper, I briefly note a few opportunities of this kind.

Reuse. Viewpoints, being system-independent, are highly reusable. The viewpoint construct is intended to facilitate capture of one important kind of architectural knowledge: *when to apply given representational mechanisms to address particular stakeholder concerns* [5]. However, very little of present architectural knowledge is captured in this fashion. For example, there is much work in the academic literature on modeling architectures via components, ports, connectors, roles, and their configurations which might be termed a "Structural Viewpoint." By having a clear viewpoint declaration, it would be easier to apply this knowledge more uniformly. One useful role for organizations like SEI would be to serve as a repository for reusable viewpoints.

View Checking. IEEE 1471 is essentially silent on the issue of checking or analysis of individual views, except to say that a view must be well-formed with respect to its viewpoint—delegating the checking to any technique associated with the viewpoint. Viewpoints will vary in their rigor, associated analytic techniques, etc., which may be brought to bear on checking a view. By having uniform declarations it may be possible to "lift" techniques developed for one notation to use with others. See [2] for a discussion of this in the context of use of the various notations of UML.

View Integration and Inter-view Consistency. It has been long recognized that introducing multiple views into architectural descriptions leads to an integration problem—how does one keep views consistent, non-overlapping?

> Complex specifications require structure, such as different segments for different concerns. However, different concerns also lead to different notations… [T]his leads to *a multiple-view problem*: different specifications describe different, but overlapping issues. [8] [my emphasis]

The introduction of viewpoint declarations, while not solving the problem, gives us a tool for detecting overlaps and inconsistencies, and potentially a substrate for solving the integration problem. See [3], [4], [1] for three different suggestions for tackling the view integration problem.

Formalization. The conceptual framework of IEEE 1471 is an informal, qualitative model. If it is useful, which appears to be the case, it may be insightful to attempt to formalize the concepts therein. Such a formalization could have benefits in several of the topics just mentioned: viewpoint reuse, view checking, view integration, and inter-view analysis.

Finally, there is another set of advanced topics in architectural description barely addressed by today's languages and tools. See [6] for discussion.

B.5 References

[1] Alexander Egyed and Rich Hilliard. "Architectural integration and evolution in a model world." In Bob Balzer and Henk Obbink, editors, *Proceedings Fourth International Software Architecture Workshop (ISAW-4), 4 and 5 June 2000, Limerick, Ireland*, pages 37-40, 2000.

[2] Rich Hilliard. "Using the UML for architectural description." In Robert France and Bernhard Rumpe, editors, *<UML>'99 The Unified Modeling Language, Second International Conference,* volume 1723 of *Lecture Notes in Computer Science*, pages 32-48. Springer, 1999.

[3] Rich Hilliard. "Views and viewpoints in software systems architecture." Position paper from the *First Working IFIP Conference on Software Architecture*, San Antonio, 1999.

[4] Rich Hilliard. "Views as modules." In Bob Balzer and Henk Obbink, editors, *Proceedings Fourth International Software Architecture Workshop (ISAW-4), 4 and 5 June 2000*, Limerick, Ireland, pages 7-10, 2000.

[5] Rich Hilliard. "Three models for the description of architectural knowledge: Viewpoints, styles, and patterns." Submission to WICSA-2, January 2001.

[6] Rich Hilliard and Timothy B. Rice. "Expressiveness in architecture description languages." In Jeff N. Magee and Dewayne E. Perry, editors, *Proceedings of the 3rd International Software Architecture Workshop*, pages 65-68. ACM Press, 1998. 1 and 2 November 1998, Orlando FL.

[7] IEEE. *Recommended Practice for Architectural Description of Software-Intensive Systems*, October 2000.

[8] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an emerging discipline*. Prentice Hall, 1996.

Appendix C: Constructing Blueprints for Product Line Platforms

Steve Ornburn
GBC-Group, Inc
January 9, 2001
sbo@gbc-group.com

Abstract

An enterprise IT architecture group was established for a client in the financial services industry. As part of their work, the enterprise architects have experimented with a range of approaches for representing architectural ideas, testing their approaches with a number of different audiences. Within the team, a consensus is beginning to emerge as to the boundaries of the systems to be represented, the required views, and the level of detail and notational conventions to be used in those views. The team and its customers are also reaching some agreements on the processes for collecting, validating and using architectural information.

C.1 Overview

For IT architects to succeed, they must have a role defined in the enterprise's business and IT processes, customers for deliverables, sources of information, and methods for engaging their customers and otherwise carrying out their responsibilities. Architectural drawings, i.e., blueprints, and associated written specifications are an architect's main deliverable. The drawings and specifications are valuable not only as a record of what has been decided; they are also valuable for having driven the decision making process. Constructing a blueprint and associated written specification entails a process rich in fact finding, analysis, negotiation, synthesis and consensus building.

Architecture is about understanding customer needs, relevant constraints, and available design patterns; an architect generates a solution by finding a combination of patterns that meets customer needs subject to the relevant constraints.[1] A perfect solution is never possible, so architecture is inevitably about making tradeoffs. In generating and evaluating alternatives, architects must consider the solution from a number of points of view. From the field of systems engineering, architects often view solutions from functional, physical, opera-

1 Christopher Alexander, *The Timeless Way of Building*, Oxford University Press, 1979.

tional, and user interface perspectives.[2] Views often useful to software architects are described by Krutchen.[3] Yet other approaches to modeling architectures derived from Zachman's work.[4,5] All are good, but serve different purposes. In my practice, I often use blueprints to bring into view overlooked operational or physical requirements. These frequently overlooked requirements are important and must be considered when making trade-offs. If they are not considered, the resulting architectures will be sub-optimal.

A couple of years ago a client asked me to help establish a group of enterprise architects to be responsible for the long-term technical evolution of the IT infrastructure for a financial services company. Establishing a group of enterprise architects has been part of a larger initiative to increase the maturity of the IT department's development processes. For this IT department, the critical problem has been that of making the transition from a mainframe shop responsible for maintaining financial records to a new role as a full partner in developing and introducing new products and services. Behind this change in role was the reality that IT technologies had become central elements in all new customer offerings, i.e., IT systems had become components in product-line architectures.

Because of the department's new role, it became important for IT managers and technical staff to define platforms supporting the company's various product and service offerings. To this end, the company's enterprise architects use architectural drawings to identify the boundaries and describe the structure of those IT platforms. The increased attention to IT platforms and product-line architectures is pivotal: heretofore, the IT department's attention had centered on projects, resulting in a siloed environment in which each project had its own, independently developed requirements, technical design, testing, change management, and delivery schedule. Now, the enterprise architects are introducing a new model, in which IT platforms are envisioned as part of a product-line architecture and managed through a process and released on a schedule. In this new model, projects are viewed as incremental changes to existing platforms and must be managed as part of that platform's release process. Architects, in this new model, are responsible for the technical design of the platform. The three-way conversation among architects, business analysts, and technical leads is inevitably a negotiation in which the architectural vision is aligned, on the one hand, with long-term business needs and, on the other, with shorter-term project needs. The enterprise architect engages throughout a project to maintain these alignments. Enterprise architects are involved in project initiation; change control, particularly when architectural trade-offs are at issue; and in preliminary and critical design reviews. Enterprise architects also participate in the testing

---

2 Dennis M. Buede, *The Engineering Design of Systems*, Wiley, 2000.

3 Philippe Kruchten, "Architectural Blue Prints—the 4+1 View Model of Software Architecture", *IEEE Software*, 12(6), Nov., 1995, pp 42-50.

4 Bernard H. Boar, Chapter 2, *Constructing Blueprints for Enterprise IT Architectures*, Wiley, 1999.

5 Steven H. Spewak, *Enterprise Architecture Planning*, Wiley, 1992.

and acceptance of project deliverables into release. verifying the architectural qualities have been suitably addressed.[6]

The importance of the enterprise architect will continue to grow as the IT platforms are re-architected to include more purchased software packages; to integrate with service bureaus and application and network service providers; and to allow for more subcontracted development. Each of these changes places an increased burden on the IT department to define long-term platform strategies, understand platform-level requirements and constraints, and accurately analyze trade-offs. Failure to carry this burden can result in architectural mismatches, unmet expectations, a high-rate of requirements churn, slipped delivery dates and overall low-quality solutions. While these types of problems are not new to IT projects, they become more difficult and more expensive to fix when they involve contractual relationships.

For an enterprise architect, blueprints and specifications are key tools in defining and managing an IT platform. For enterprise architects, identifying platform boundaries, describing platform structure, and identifying key platform qualities have all required significant data collection and reverse engineering. Users and business analysts describe operations and how they are embedded in business process. Software engineers tell us the sequence of steps required to carry out an operation, naming the components involved and describing the connectivity. Typically, software engineers will also identify the network nodes to which components are deployed. Frequently, we observe mismatches between the operations identified by users and those discussed by software engineers, with software engineers seeing a finer-grained set of functions and not seeing how users apply those functions to get work done. Network team sees physical nodes and their connectivity at a low level. Data team provides a rich understanding of the enterprise data but often cannot describe what the data is used for or how it is processed.

Initially, enterprise architects recorded architectural data in simple box and arrow diagrams. The early blueprints revealed at least as many styles and conventions as there were architects. As architects gained experience drawing, explaining, and reading blueprints, and as they better understood the platforms they were characterizing, their drawing styles began to converge. Architects became more selective of details recorded about specific components. Architects also adopted more sophisticated techniques for encoding information about relationships among those components. Furthermore, with experience, the enterprise architects became more attentive to matters such as notational standards, traceability, change control, and independent reviews.

Enterprise architects are now making a transition from documenting "as built" architectures to evaluating those architectures on the basis of their documentation. In the course of these evaluations, the architects have identified several important architectural qualities that have

---

6 The architectural practices continue to evolve in a process of mutual adaptation. The enterprise architects select and adapt to industry best practices and then, with experience, modify those practices and make them their own.

been neglected in one or more platforms. Examples of points raised in architectural reviews include:

- Given the high rate of technical innovation, architectures must be extensible, permitting the addition of new capabilities not foreseen at the time of the original design. Some of the architectures reviewed have achieved extensibility through the use of layered architectures, the façade design pattern, and loosely coupled components interconnected through a message broker.
- Given business uncertainties, particularly around customer and user willingness to adapt to new information systems, it must be possible to introduce small, exploratory systems which can then be enhanced and scaled as required. In a distributed environment it should be possible to incrementally scale a system by adding additional processing nodes along with mechanisms for fail over and load balancing.
- Architectural blueprints have been the input to several mathematical models or simulations evaluating system performance, including capacity, throughput, and response times. The relationships between volumes and response times have been forecast, and these results in turn have driven capacity planning.
- Given the number of third parties who may have occasion to modify or enhance systems, maintainability is becoming a business-critical architectural quality. To enhance maintainability, attention must be given to selecting and following well-known design patterns, accurate and readable documentation, particularly for architectures and high-level designs, requirements traceability, and effective configuration, requirements, and change management.
- By including operational procedures in business models, enterprise architects have been able to ensure that those procedures and associated support systems have been designed in a way that is consistent with availability and reliability requirements.
- Architectural review of behavioral models, while not looking at overall system correctness, has been able to ensure that key safety, liveness, and fairness properties have been satisfied. For example, architectural review detected potential race conditions—e.g., on system initialization it was possible for one thread to read a data structure before another had initialized it.

We have found that for these reviews, the most useful artifact is a blueprint providing a composite view of the architecture. This composite view combines structural information often found in separate logical (application), component, and deployment views. The composite structural view also allows architects to express relationships by overlaying business, technology, and information architectures. While behavioral descriptions of the system, e.g., sequence diagrams or state charts, are shown separately, they can easily be related back to the composite structural view.

Some new readers have said that the density with which information is packed onto a blueprint can make composite structural views hard to understand. However, experienced readers (including one sr. vice president from the IT department) seem to prefer the convenience of a single reference diagram. Experienced readers also have found the overlaying of information from multiple views helps show how the parts make the whole.

C.2 Business Layer

The business layer shows the touch points between business processes and IT systems. Activity at touch points can be included as part of use case. Touch points are generally represented with an icon for user or device. The business layer also identifies the applications that can be accessed through the touch points. Applications are represented as shaded boxes into which technology components can be placed. Sometimes an application may be divided into components representing different capabilities or processing steps. Enterprise architects have extended the concept of application to include processes and services related to the operation and maintenance of IT systems. Architects made this extension after discovering cases for operational requirements and procedures were not considered during system design, resulting in solutions that could not be upgraded without significant customer impact.

C.3 Technology Layer

Technology components are represented as white boxes placed within the shaded areas representing applications. Typically, in a distributed environment, a technology component is a server and the software and hardware components installed on it. The text within the white box identifies the software and hardware components deployed on that server. If the components work together to provide a service, some notation may be made to represent their collaboration. The list of software and hardware can include system software, hardware platform, lower-level services and frameworks used, development environment, hardware platform and other resources needed for the application. When the relationships among technology components are many and complex, an analysis diagram detailing those relationships may be provided. It may also be useful to provide "cross-sectional" views of the component by describing its internal structure and behavior. These cross sections may be provided either as insets on the main diagram (space permitting) or on a separate page.

Connectors, represented as lines on the structural view, are mechanisms for transferring control or information from one technology component to another. While represented as lines rather than boxes, connectors are nevertheless considered first-class components, and may be associated with attributes or a cross-sectional view. Enterprise architects have characterized a variety of connectors, including some embedded in other multi-function components, some implemented as separate pieces of software, and yet others available to developers as features in their programming languages. Architects have also characterized connectors employing extensive object-relational mapping using tools such as TopLink and Persistence Builder. Yet another connector characterized by architects used the façade design pattern to maintain extensibility and promote reuse.

C.4 Information Layer

At the information layer, components typically identify distinct subject matters. Cross sections can be used to show details about the information model. The cross section may make of any of a variety of information modeling techniques including object models and ER diagrams. In some cases the architect may choose to describe how the information model has been optimized as for use in data mart.

C.5 Behavioral View

Structural views generally must be accompanied by some description of how the components work together to effect some result or behavior. Enterprise architects frequently use sequence diagrams to characterize behavior. Generally, architects are not interested in complete behavioral specifications. Instead, they characterize key fairness, liveness and safety the system must satisfy. For example, a safety constraint might have the form "when component x fails, it must fail in a way that does not create a security hole."

Every system includes logic for generating the behavior. In some cases the logic for deciding what to implement next is based on an architecturally interesting coordination mechanism. Some systems use a distinct workflow engine to decide what to do. Other systems employ a coordination mechanism based on publish/subscribe messaging. There are many other types of coordination mechanisms that may be used. When the coordination mechanism is architecturally interesting, its structure and behavior can be described by means of cross-sectional views.

C.6 Recent Adaptations

Enterprise architects continue to adapt their conventions for representing architectures. The systems they work with are heterogeneous, but at the top level, systems can be conveniently viewed as having an n-tiered, client server structure. This structure is now being used to control the placement of technical components; each component is classified as belonging to presentation interfaces, middle tier presentation services, middle tier business logic, workflow control, enterprise business logic, enterprise data, and enterprise data movement. Some multi-function components may span more than one tier. Architects have generalized the notions of the tiers to ensure that various systems can be classified.

Naturally Web browsers are classified as presentation interface, and fat clients span both presentation interface and presentation services. The mailroom, responsible for scanning incoming mail and placing in workflow queues, is presentation. The component containing the logic for putting the document image on the workflow queue is itself classified as workflow. The menuing system on the IVR spans presentation and presentation services. Other components on the IVR are classified as middle tier business logic. The Geotel system for call routing and computer telephony integration is classified as a workflow engine. Record keeping systems are enterprise business logic and the associated data stores are enterprise data. Off site storage of documents is also enterprise data. In some cases, semi-automated record keeping is enterprise business logic, with people and their business processes encapsulated within a technology component.

Appendix D: DREAM Framework A Context for System Development

Tony Thompson
Information Technology
Warner Music Group
Burbank, CA 91505

Abstract

The Web has transitioned enterprises from an informal data system initiative to a content-rich, data-driven distributed system with a mission-critical imperative. A descriptive requirements engineering attribute method (DREAM) framework is presented as one tool in an architectural "integrate and conquer" strategy towards rapid, reliable system development and deployment. As the need for architectural description becomes critical, a "lighter-weight" framework is proposed as a philosophical asset within the overall system lifecycle process. Within this context, the production of several documentation artifacts is needed as the organization manages the project, provides the necessary development infrastructure and training for project members, and improves the project's software process. Done properly, organizations/companies will have the opportunity to reuse inventories of code, which will make it faster and easier for the deployment of new Web Applications and data-driven Web sites.

D.1 Problem-Statement

With the lure of growing their market faster, the creation of systems more cost effectively, and utilization of open source code centric approaches, companies are revolutionizing the software industry and enterprise systems. The migration to standards-based applications creates a variegated component-dependent architecture, affecting how the systems are designed, developed and deployed.

A specific component brings a specific implementation-dependent application package. Package viability, albeit vendors for commercial packages and project participants for open-source, introduces an unmatched form of investment risk exposure, which must be managed from the inception phase of the system. A technical challenge is also faced during the migration of an existing application to a competing product judged to be of superior constitution.

Setting aside any contractual considerations, the similarity of the described conditions mandates that requirements management, design knowledge capture, and interface and architectural attributes are well formed and sufficiently described throughout the system life-cycle.

D.2 Approach

The DREAM framework draws necessarily upon predecessor life-cycle process systems (such as ISO/IEC 12207). DREAM differs as it challenges the common knowledge that requirements are descriptions of what the software does, while the design describes how the software does it. Rather DREAM introduces the paradigmatic shift that the design is a configuration item that is merely an attribute of the architecture. The configuration item might utilize more than one piece of technology to meet its obligation. Technology is traded-off as a dimension against other architectural dimensions:

Application Architecture which defines the structure and relationships of the software that embodies the business processes

Development Architecture which defines the environment of tools, methods, processes, and team structures to develop and maintain systems

Data Architecture which defines the structure and models of the information assets of the organization, principally databases, but also digital content

Although one might argue that this might be some semantic slight-of-hand, the framework is not at all oriented towards the rigidity seen in previous standards (vis-à-vis MIL-STD-2167). The framework rather extends a pattern witnessed in human hierarchical systems. It allocates to a design configuration item the accountability, responsibility, and authority for performance of those assigned architectural attributes. Within a behavioral context, the architectural attributes equate to the assigned system tasks of that particular element.

Allegorically, this is akin to building a wall by ensuring tight fit between cut stones rather than by uniform brick. Unlike previous life-cycle systems, the details of "how to" perform the activities and tasks included in the configuration item are mandatory. The implementation requirements are then attributed to the package and integral components, as a direct consequence of their intrinsic capabilities. Behavior is dynamic, and can be more readily associated with so-called quality factors, like performance, which are difficult to capture in UML.

By adapting material from the Windows DNA team, we can formally state what characteristics the documentation artifacts must have to be of greatest utility to a specific project. Some of the most common causes of failed deliveries of systems are:

Dissociation between the developer and the business decision-maker – Lack of visibility into the status of the project is a significant issue. The documentation artifacts should provide for succinct resolution of assigned attributes at the time of inspection. Requirements traced to these attributes will correctly provide assessment of feature incorporation, and should ideally be uniquely identifiable. A primary process problem is that use cases come later than project start.

Insufficient testing and quality control – Conditions related to incompleteness in attribute assignment should be easily assessable. The translation of use case material into the architectural elements should easily provide for optimal feature test cases, which ARE contained in the artifact.

Inadequate requirement management – Lack of skilled capture of business requirements tends to result in the all too familiar pattern of feature creep. Decisions that resulted in specific architectural assignments should be visible in the artifacts, as should the rationale for a non-implementation decision.

Allocating complex processes into a single system element – Feature clustering is an emerging threat to complexity management in the architecture. Single package or component overload must be discernable, to assure either reallocation (preferred) or specific risk identification (acceptable).

Premature component technology changeout –The battle to have the "latest and greatest" and the "best of breed" prior to fully scoping the attributes and associated requirements results in suboptimal migration. Although not entirely in the purview of documentation, the artifacts should be "exposure friendly", such that the feature set is exposed, even if the allocation statement designates an unused condition. An unused condition provides objective evidence that the particular feature should not be a driver in product evaluation.

Technology, Applications Development and Data architectures are considered viable and complementary architectural perspectives. Within the UML oriented tools, the notation for assignment into architectural elements is reaching standardization, but has not reached a normalized condition, and is without a concurrent documentation artifact standard. The most crucial aspect of system architecture is the component selection. System architecture is challenging. Allocating architecture attributes based on associated OSI tiers and commercial/open source package capabilities helps to reduce the complexity of the architecture task.

D.3 Suggested artifacts

With an eye on tailoring, the specific documentation should be adjusted as necessary to scale within the magnitude of the project. This is not inclusive since more plans, records and reports would conceivably be furnished on an as needed basis. The larger the project, the more complex the document. Smaller projects can consolidate documents as appropriate. All of these artifacts must be considered living documents. No formats are presented due to variability between projects, although acronym correspondence with IEEE/EIA 12207 is deliberate.

1. Project Approach Document (PAD) – Project scope, organization structure, key milestones and list of related projects and organizations. Similar to a basic statement of work, the PAD provides the basis for detailed tracking by allocating named personnel to packages as part of the work breakdown structure.
2. Operation Concept Document (OCD) – Provides the content of a standard OCD, but supports the incorporation of business user requirements, and may be broken into two support documents.
3. Functional Requirements Document (FRD) – The business users view of the system. Should have a one-to-one, but typically one-to-many relationship with lower level requirements. The document would contain use-cases, and relate to any interactive material in UID.
4. User Interface Document (UID) – The requisite business system screens and general report requirements.
5. System Requirements Document (SRD) – Provides the high-level association between the system requirements and the system components, thus defining the architecture. Additional use cases are provided to indicate interaction between architectural elements. Static functions are assigned as attributes, along with performance concerns. Otherwise similar to the SSDD from 2167.
6. Software Design Document (SDD) – Provides both the assigned attributes, test requirements, and details of the significant objects and methods used. Describes key functions of the hosting application environment, as applicable, for example specific method calls to an application server. Changes any time the software changes.
7. Database Design Document (DDD) – Provides standard database specifications, and includes physical implementation, including stored procedures and triggers.
8. Test and Configuration Management Plan (TCMP) – Provides one-stop physical and process requirements for the management of tests, versions, and environments. Combines elements typically found in planning documents entitled Development process, Maintenance process, Operation process, Software CM, Software integration, and the Test or validation plan.
9. System Training Manual (STM) – Provides information for user training and operations.

D.4 Progress

With systems being built by both employees and consultants, uniformity of format and content have been decided on a per project basis. Future projects are being mandated for greater compliance with management endorsement, and project documentation is utilizing a standard template and content standard. It is hoped that these inputs will contribute to establishing baseline for the documentation artifacts.

D.5 Conclusion

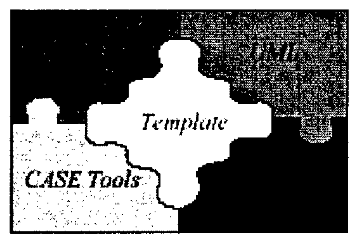
The DREAM framework is provided as a meaningful excursion into the philosophy of system architecture, requirements engineering, and product realization. Architectural tradeoffs are an essential activity for system implementation. It ensures system compliance and system suitability for its intended function. Each package involved in the system goes through rigorous iteration within the context of an architectural element. This iteration includes conceptual build and testing to ensure package compatibility with assigned architectural attributes. Early implementation enhances system reliability through capability assurance, which creates customer confidence. The document artifacts lend themselves to visibility into the software lifecycle process.

Appendix E: Position on Software Architecture Documentation

Jeff Tyree. Enterprise Architect
CapitalOne, Richmond VA
804-934-8961
jeff.tyree@capitalone.com

E.1 Introduction

As an architect for CapitalOne, I have spent quite some time over the last few months thinking about the question "How should architecture be documenting?" As the glyph[1] to the right suggests, architecture description is interwoven with other aspects of architecture construction. This *Position on Software Architecture Documentation* will discuss the principles that architecture must follow, qualities that it must possess and good examples to use for discussion.



E.2 Principles

Supports the Shared Vision: The architecture description must be determined in such a way as to be consistent with how one's organization defines architecture. If the organization sees architecture's role as one of providing guidelines and standards for coding and product selection, the description needs to address these concerns. If the organization sees the role as the primary driver for how systems are built (e.g., system structuring), the description needs to meet this need. There can be no disconnect between the organization and an architect(s) on the vision. The implications associated with this principle are far reaching. If an architect wishes to stretch the boundaries of how he/she operates, the vision is the first place to start. A counter-argument to this approach is for architects to construct their architectural vision and then win over their peers. A difficult, if not impossible, task.

[1] Tutorial UML World 2000, Coleman, http://www.architecture.external.hp.com/

E.2.1 Supports the Communication Channels

The architecture description must be determined in such as way as to meet the various stakeholders' needs. It needs to be described in such a way as to manage outward (e.g., business, managers) and manage inward (developers. testers). There needs to be organizational standards that are agreed upon in order for proper communication to take place. If UML, RM-ODP, etc. are used then all stakeholders (including peers) need to understand the language used to communicate. The implication associated with this principle is that the architect cannot choose languages (e.g., ADLs) indiscriminately. He/she needs to consider the audience. Consider Janis Putman's comments in her well-written text on RM-ODP:

> *Be careful in the selection of a modeling tool for RM-ODP concepts. The object modeling concepts of RM-ODP are different from those of most modeling tools. One will need to map the concepts used from RM-ODP to those of the tool selected*[2].

MCI Systemhouse authored a 40-page paper describing the mapping between RM-ODP and UML[3]. A counter-argument to this approach is that who ever constructs an artifact should be allowed to choose how (and in what language) it is constructed. Good luck to those on this path.

E.2.2. Must Integrate with Supporting Tools

Architecture description must be more than Write-Once, Read-Once. One required property of architecture description is that it must be enduring. As the architecture evolves, so must its description. The implication of this principle is that it must be straightforward to keep the description in-sync with the thing it is describing. As engineers are adverse to documentation-driven pain, this process must be as automated as possible. CASE tools, templates, word processing and configuration management tools must be integrated to ensure that the architecture description does not die on the vine. Although the RM-ODP is compelling as a standard, the lack of tool support is even a more compelling reason to not adopt.

A counter-argument is given by Janis Putman who states

> *The use of a tool never replaces the needed system engineering, system analysis, software analysis, or architecture engineering of the system. Tools are always limited.*[4]

This, of course, is true. Which path you take depends on which you believe to be *more true*.

---

2 Architecting with RM-ODP, Janis R. Putman, 2000

3 Relationship of the Unified Model Language with the Reference Model of Open Distributed Computing, MCI Systemhouse, 1997.

4 Architecting with RM-ODP, Janis Putman, Chapter 4.

E.2.3 Must Support the Architectural Process

The architecture description must conform to the process used to construct it. The template used for description isn't a process, but is a starting point for a deliverable. A template should guide engineers in creating designs that maintain the integrity of the architecture. For example, if architectural styles are to be used in the description of the architecture (determined by a step in the process), the template used for description needs to give guidelines as to how styles are documented and conveyed.

E.3 Qualities

The qualities of good software architecture documentation mimic the qualities of good architecture.

E.3.1 Is Usable

The architecture description needs to support the seven rules of good software organization[5]. These include:

1. *Written from the viewpoint of the reader*: This implies that multiple viewpoints are needed in order to support various levels of abstraction. It should also written for ease of reference, not ease of reading. This aspect promotes read-many versus read-once.
2. *Avoids repetition.* Don't you hate it when you read an architecture document that is a rehash of the requirements document you just reviewed?
3. *Uses a standard organization.* (See Supports the Communication Channel.)
4. *Records rationale.* (See Is Defensible.)
5. *Avoids ambiguity.* (See Is Actionable and Is Testable.)
6. *Remains current.* How many times have you reviewed a document to later find out it was obsolete?
7. *Fits its Purpose.* This is closely related to the first rule.

E.3.2 Is Actionable (or prescriptive)

Architecture must be described to the level of detail to support its construction. If architecture is used to partition work, where components are to be constructed by developers, the architecture needs to describe the interfaces and semantics to a sufficient level of detail to minimize integration issues, communications paths, etc.

[5] Software Architecture Documentation in Practice, see www.sei.cmu.edu site.

E.3.3 Is Testable (or precise)

An architecture needs to be testable. A box and line diagram does not an architecture make. A level of precision is required in order to support reasoning about the architecture. The level of precision necessary depends on the validation method.

E.3.4 Is Defensible

The architecture documentation must show how the system supports the quality aspects. These requirements may conflict and there may be trade-offs among competing concerns. The documentation must clearly illustrate the principles, constraints and rationale for choices made. How many times has there been a change in architects with the new architect making fundamental changes to the system structure? I believe that one of the main reasons for this *churn* is that decisions were not clearly articulated and the vision not totally assimilated.

E.4 Good Examples

E.4.1 HP's Architecture Template

The best example that I've seen for describing architecture is HP's Architecture Template[6]. This template has the advantages over others that I've seen, including Rational's, IBM's, MITRE's and our internal Blueprints. It stands out from the others in the following ways:

- It is integrated with available tools, such as Rational Rose and SoDA.
- It supports a lightweight process.
- It focuses on component models versus logical design models.
- It is a starting point for providing a "teaching" template.
- It acknowledges the fact that Architecture Documents may be *overviews* or *reference manuals* and provides for both types and the evolution from *overview* to *reference.*
- The section on component interactions is very good. Coleman keeps true to his Fusion roots and provides for Component Interaction Models. These models were the best aspects of the Fusion Process.
- The acknowledgement of the importance of Mechanisms is significant. When constructing an architecture document for a recent project, I explicitly included a section on Mechanisms and in many ways it was the most descriptive of the sections I created.
- It provides a good description of meta-architecture.

6 www.architecture.external.hp.com

E.4.2 Bredemeyer's Action Guides

Dana Bredemeyer and Ruth Mulan[7] formerly of HP, have done some excellent work in extending the HP process and templates. They have provided useful action guides for several key process items, including capturing stakeholder needs, principles, and a context map.

E.4.3 Grove Graphics Graphic Guides

The Grove Consultants International[7] firm has produced some useful Graphics Guides for strategic visioning. It is nice to see that others hold the view that a vision is not a 50-page document.

E.4.4 Applied Software Architecture.

The *Applied Software Architecture*[8] text provides an interesting section on Global Analysis along with a useful way to document issues and strategies.

E.4.5 Software Architecture Documentation in Practice[9]

The beginnings of this text are very promising.

E.4.6 RM-ODP Standard[10]

The RM-ODP is the definitive standard for describing distributed architectures. Its descriptions are formal, precise and complete. Its coverage of distribution transparencies and functions is very thorough. (I like its choice of views as they mimic the skill sets of architects that would be responsible for their construction.)

E.4.7 State of North Carolina[11]

The State of North Carolina has done a very good job in describing the meta-architecture. Their descriptions of principles, guidelines, best practices and technology component discussions are some of the best documented that I've seen. Unlike other sites, the rationale for their choices is clear and concise.

---

7 http://www.grove.com/services/tool_guides.html

8 Applied Software Architecture, Hofmeister, Nord, Soni

9 See www.sei.cmu.edu

10 See www.iso.ch

11 See http://irm.state.nc.us/techarch/archfrm.htm

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE MAY 2001	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE SEI Workshop on Software Architecture Representation, 16-17 January, 2001		5. FUNDING NUMBERS F19628-00-C-0003
6. AUTHOR(S) Felix Bachmann, Paul Clements, David Garlan, James Ivers, Reed Little, Robert Nord, Judy Stafford		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2001-SR-010
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12B DISTRIBUTION CODE
13. ABSTRACT (MAXIMUM 200 WORDS) To further its work in architecture-related ideas, the SEI held its first Architecture Representation Workshop, 16-17, January, 2001. Five leading software architects and practitioners were invited to discuss aspects of the architecture representation with senior members of the SEI technical staff. The workshop articulated best practices, identified gaps in the available technology, and set the direction for future efforts.		
14. SUBJECT TERMS architecture representation, documentation, use cases, architecture descriptions		15. NUMBER OF PAGES 64
16. PRICE CODE		

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL